Άσκηση 1: Απλή Εφαρμογή του Dijkstra

Περιγραφή: Δημιούργησε ένα γράφο με τέσσερις κόμβους. Οι κόμβοι είναι συνδεδεμένοι μεταξύ τους με διάφορα βάρη. Χρησιμοποίησε τον Αλγόριθμο του Dijkstra για να βρεις την πιο σύντομη διαδρομή από τον κόμβο 1 σε όλους τους άλλους κόμβους και εμφάνισε τις αποστάσεις.

Λύση:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Δημιουργία του γράφου
        var graph = new Dictionary<int, List<Tuple<int, int>>>
        {
            { 1, new List<Tuple<int, int>> { Tuple.Create(2, 1), Tuple.Create(3,
4) } },
            { 2, new List<Tuple<int, int>> { Tuple.Create(3, 2), Tuple.Create(4,
5) } },
            { 3, new List<Tuple<int, int>> { Tuple.Create(4, 1) } },
            { 4, new List<Tuple<int, int>>() }
        };

        // Εφαρμογή του αλγορίθμου Dijkstra
        var distances = Dijkstra(graph, 1);

        // Εκτύπωση των αποτελεσμάτων
        foreach (var kvp in distances)
        {
            Console.WriteLine($"Απόσταση από 1 στο {kvp.Key}
είναι {kvp.Value}");
        }
    }

    static Dictionary<int, int> Dijkstra(Dictionary<int, List<Tuple<int, int>>>
graph, int start)
    {
        var distances = new Dictionary<int, int>();
```

```
        var priorityQueue = new SortedSet<Tuple<int, int>>(Comparer<Tuple<int,
int>>.Create((a, b) => a.Item2.CompareTo(b.Item2)));

        foreach (var node in graph.Keys)
        {
            distances[node] = int.MaxValue;
        }
        distances[start] = 0;

        priorityQueue.Add(Tuple.Create(start, 0));

        while (priorityQueue.Count > 0)
        {
            var current = priorityQueue.Min;
            priorityQueue.Remove(current);

            int currentNode = current.Item1;
            int currentDistance = current.Item2;

            if (currentDistance > distances[currentNode])
            {
                continue;
            }

            foreach (var neighbor in graph[currentNode])
            {
                int neighborNode = neighbor.Item1;
                int edgeWeight = neighbor.Item2;
                int newDistance = currentDistance + edgeWeight;

                if (newDistance < distances[neighborNode])
                {
                    distances[neighborNode] = newDistance;
                    priorityQueue.Add(Tuple.Create(neighborNode, newDistance));
                }
            }
        }

        return distances;
```

```
    }
}
```

Άσκηση 2: Βελτίωση του Δικτύου Μεγαλύτερου Γράφου

Περιγραφή: Δημιούργησε έναν πιο σύνθετο γράφο με 6 κόμβους. Εφαρμόστε τον Αλγόριθμο του Dijkstra για να βρείτε την πιο σύντομη διαδρομή από τον κόμβο 1 σε όλους τους άλλους κόμβους. Εμφάνισε τις αποστάσεις και επιβεβαίωσε ότι τα αποτελέσματα είναι σωστά.

Λύση:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Δημιουργία του γράφου
        var graph = new Dictionary<int, List<Tuple<int, int>>>
        {
            { 1, new List<Tuple<int, int>> { Tuple.Create(2, 7), Tuple.Create(3,
9), Tuple.Create(6, 14) } },
            { 2, new List<Tuple<int, int>> { Tuple.Create(1, 7), Tuple.Create(3,
10), Tuple.Create(4, 15) } },
            { 3, new List<Tuple<int, int>> { Tuple.Create(1, 9), Tuple.Create(2,
10), Tuple.Create(4, 11), Tuple.Create(6, 2) } },
            { 4, new List<Tuple<int, int>> { Tuple.Create(2, 15),
Tuple.Create(3, 11), Tuple.Create(5, 6) } },
            { 5, new List<Tuple<int, int>> { Tuple.Create(4, 6), Tuple.Create(6,
9) } },
            { 6, new List<Tuple<int, int>> { Tuple.Create(1, 14),
Tuple.Create(3, 2), Tuple.Create(5, 9) } }
        };

        // Εφαρμογή του αλγορίθμου Dijkstra
```

```
        var distances = Dijkstra(graph, 1);

        // Εκτύπωση των αποτελεσμάτων
        foreach (var kvp in distances)
        {
            Console.WriteLine($"Απόσταση από 1 στο {kvp.Key} είναι {kvp.Value}");
        }
    }

    static Dictionary<int, int> Dijkstra(Dictionary<int, List<Tuple<int, int>>> graph, int start)
    {
        var distances = new Dictionary<int, int>();
        var priorityQueue = new SortedSet<Tuple<int, int>>(Comparer<Tuple<int, int>>.Create((a, b) => a.Item2.CompareTo(b.Item2)));

        foreach (var node in graph.Keys)
        {
            distances[node] = int.MaxValue;
        }
        distances[start] = 0;

        priorityQueue.Add(Tuple.Create(start, 0));

        while (priorityQueue.Count > 0)
        {
            var current = priorityQueue.Min;
            priorityQueue.Remove(current);

            int currentNode = current.Item1;
            int currentDistance = current.Item2;

            if (currentDistance > distances[currentNode])
            {
                continue;
            }

            foreach (var neighbor in graph[currentNode])
            {
```

```
                int neighborNode = neighbor.Item1;
                int edgeWeight = neighbor.Item2;
                int newDistance = currentDistance + edgeWeight;

                if (newDistance < distances[neighborNode])
                {
                    distances[neighborNode] = newDistance;
                    priorityQueue.Add(Tuple.Create(neighborNode, newDistance));
                }
            }
        }

        return distances;
    }
}
```

Άσκηση 3: Αναγνώριση Μη Ευκτής Διαδρομής

Περιγραφή: Δημιούργησε έναν γράφο με 5 κόμβους, όπου ορισμένοι κόμβοι δεν συνδέονται μεταξύ τους. Εφαρμόστε τον Αλγόριθμο του Dijkstra για να βρείτε την πιο σύντομη διαδρομή από τον κόμβο 1 σε όλους τους άλλους κόμβους. Εμφάνισε τις αποστάσεις και βεβαιώσου ότι οι αποστάσεις των μη συνδεδεμένων κόμβων δείχνουν σωστά (π.χ., int.MaxValue).

Λύση:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Δημιουργία του γράφου
        var graph = new Dictionary<int, List<Tuple<int, int>>>
        {
            { 1, new List<Tuple<int, int>> { Tuple.Create(2, 2), Tuple.Create(3,
4) } },
            { 2, new List<Tuple<int, int>> { Tuple.Create(4, 7) } },
            { 3, new List<Tuple<int, int>> { Tuple.Create(4, 1) } },
```

```
            { 4, new List<Tuple<int, int>>() },
            { 5, new List<Tuple<int, int>>() } // Απομονωμένος κόμβος
        };

        // Εφαρμογή του αλγορίθμου Dijkstra
        var distances = Dijkstra(graph, 1);

        // Εκτύπωση των αποτελεσμάτων
        foreach (var kvp in distances)
        {
            Console.WriteLine($"Απόσταση από 1 στο {kvp.Key} είναι {(kvp.Value == int.MaxValue ? "Αναγνωρίσιμος" : kvp.Value.ToString())}");
        }
    }

    static Dictionary<int, int> Dijkstra(Dictionary<int, List<Tuple<int, int>>> graph, int start)
    {
        var distances = new Dictionary<int, int>();
        var priorityQueue = new SortedSet<Tuple<int, int>>(Comparer<Tuple<int, int>>.Create((a, b) => a.Item2.CompareTo(b.Item2)));

        foreach (var node in graph.Keys)
        {
            distances[node] = int.MaxValue;
        }
        distances[start] = 0;

        priorityQueue.Add(Tuple.Create(start, 0));

        while (priorityQueue.Count > 0)
        {
            var current = priorityQueue.Min;
            priorityQueue.Remove(current);

            int currentNode = current.Item1;
            int currentDistance = current.Item2;
```

```
            if (currentDistance > distances[currentNode])
            {
                continue;
            }

            foreach (var neighbor in graph[currentNode])
            {
                int neighborNode = neighbor.Item1;
                int edgeWeight = neighbor.Item2;
                int newDistance = currentDistance + edgeWeight;

                if (newDistance < distances[neighborNode])
                {
                    distances[neighborNode] = newDistance;
                    priorityQueue.Add(Tuple.Create(neighborNode, newDistance));
                }
            }
        }

        return distances;
    }
}
```

Εξήγηση

1. **Άσκηση 1:** Δημιουργούμε ένα μικρό γράφο με 4 κόμβους και εφαρμόζουμε τον αλγόριθμο για να βρούμε τις αποστάσεις από τον κόμβο 1.
2. **Άσκηση 2:** Δημιουργούμε έναν μεγαλύτερο γράφο με 6 κόμβους και επαναλαμβάνουμε τη διαδικασία για να βρούμε τις αποστάσεις.
3. **Άσκηση 3:** Δημιουργούμε έναν γράφο με απομονωμένους κόμβους για να δούμε πώς ο αλγόριθμος διαχειρίζεται μη συνδεδεμένους κόμβους.

Αυτές οι ασκήσεις θα σε βοηθήσουν να κατανοήσεις πώς λειτουργεί ο Αλγόριθμος του Dijkstra και πώς να τον εφαρμόσεις στην C#. Αν έχεις οποιεσδήποτε ερωτήσεις ή χρειάζεσαι επιπλέον βοήθεια, μη διστάσεις να ρωτήσεις!